Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящая курсовая работа выполнена в соответствии с требованиями ГОСТ Единой системы программной документации (ЕСПД) [1]. Все этапы решения задач курсовой работы фиксированы, соответствуют требованиям, приведенным в методическом пособии для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [2-3] и методике разработки объектно-ориентированных программ [4-6].

Механизм сигналов и обработчиков является одним из способов организации взаимодействия между объектами в программировании.Он позволяет установить связь между сигналами, которые генерируются объектом, и обработчиками, которые реагируют на эти сигналы. Вместе с передачей сигнала также передаются определенные данные, что делает механизм сигналов и обработчиков мощным инструментом для реализации сложных схем взаимодействия объектов.

В данной работе мы рассмотрим задачу организации взаимосвязи объектов посредством механизма сигналов и обработчиков. Мы опишем алгоритм и метод решения этой задачи, а также представим пример кода, демонстрирующего работу данного механизма. Основная цель работы - исследовать и объяснить принципы работы механизма сигналов и обработчиков, а также продемонстрировать его применение на практике.

В ходе работы мы описываем три основных метода, необходимых для организации взаимосвязи по механизму сигналов и обработчиков: метод установки связи, метод удаления связи и метод выдачи сигнала. Мы предлагаем использование указателей на методы сигнала и обработчика, что позволяет гибко

устанавливать и разрывать связи между объектами.Также мы предлагаем использовать структуру для хранения установленных связей и вектор для управления ими.

1 ПОСТАНОВКА ЗАДАЧИ

Реализовать механизм взаимодействия объектов с использованием сигналов и обработчиков, с передачей вместе сигналом текстового сообщения (строковой переменной).

Для организации взаимосвязи по механизму сигналов и обработчиков в базовый класс добавить три метода:

- установления связи между сигналом текущего объекта и обработчиком целевого объекта;
- удаления (разрыва) связи между сигналом текущего объекта и обработчиком целевого объекта;
- выдачи сигнала от текущего объекта с передачей строковой переменной. Включенный объект может выдать или обработать сигнал.

Методу установки связи передать указатель на метод сигнала текущего объекта, указатель на целевой объект и указатель на метод обработчика целевого объекта.

Методу удаления (разрыва) связи передать указатель на метод сигнала текущего объекта, указатель на целевой объект и указатель на метод обработчика целевого объекта.

Методу выдачи сигнала передать указатель на метод сигнала и строковую переменную. В данном методе реализовать алгоритм:

1. Если текущий объект отключен, то выход, иначе к пункту 2.
2. Вызов метода сигнала с передачей строковой переменной по ссылке.
3. Цикл по всем связям сигнал-обработчик текущего объекта:

    3.1. Если в очередной связи сигнал-обработчик участвует метод сигнала, переданный по параметру, то проверить готовность целевого объекта. Если целевой объект готов, то вызвать метод обработчика

целевого объекта указанной в связи и передать в качестве аргумента строковую переменную по значению.

4. Конец цикла.

Для приведения указателя на метод сигнала и на метод обработчика использовать параметризированное макроопределение препроцессора.

В базовый класс добавить метод определения абсолютной пути до текущего объекта. Этот метод возвращает абсолютный путь текущего объекта.

Состав и иерархия объектов строится посредством ввода исходных данных. Ввод организован как в версии № 3 курсовой работы. Если при построении дерева иерархии возникает ситуация дубляжа имен среди починенных у текущего головного объекта, то новый объект не создается.

Система содержит объекты шести классов с номерами: 1, 2, 3, 4, 5, 6. Классу корневого объекта соответствует номер 1. В каждом производном классе реализовать один метод сигнала и один метод обработчика.

Каждый метод сигнала с новой строки выводит:

```
Signal from «абсолютная координата объекта»
```

Каждый метод сигнала добавляет переданной по параметру строке текста номер класса принадлежности текущего объекта по форме:

```
«пробел»(class: «номер класса»)
```

Каждый метод обработчика с новой строки выводит:

```
Signal to «абсолютная координата объекта»   Text: «переданная строка»
```

Моделировать работу системы, которая выполняет следующие команды с параметрами:

- EMIT «координата объекта» «текст» – выдает сигнал от заданного по координате объекта;
- SET_CONNECT «координата объекта выдающего сигнал» «координата

целевого объекта» – устанавливает связь;

- DELETE_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» – удаляет связь;
- SET_CONDITION «координата объекта» «значение состояния» – устанавливает состояние объекта.
- END – завершает функционирование системы (выполнение программы).

Реализовать алгоритм работы системы:

- в методе построения системы:
  - построение дерева иерархии объектов согласно вводу;
  - ввод и построение множества связей сигнал-обработчик для заданных пар объектов.
- в методе отработки системы:
  - привести все объекты в состоянии готовности;
  - цикл до признака завершения ввода:
    - ввод наименования объекта и текста сообщения;
    - вызов сигнала заданного объекта и передача в качестве аргумента строковой переменной, содержащей текст сообщения.
  - конец цикла.

Допускаем, что все входные данные вводятся синтаксически корректно. Контроль корректности входных данных можно реализовать для самоконтроля работы программы. Не оговоренные, но необходимые функции и элементы классов добавляются разработчиком.

1.1 Описание входных данных

В методе построения системы.

Множество объектов, их характеристики и расположение на дереве

иерархии. Структура данных для ввода согласно изложенному в версии № 3 курсовой работы.

После ввода состава дерева иерархии построчно вводится:

```
«координата объекта выдающего сигнал» «координата целевого объекта»
```

Ввод информации для построения связей завершается строкой, которая содержит:

```
«end_of_connections»
```

В методе запуска (отработки) системы построчно вводятся множество команд в производном порядке:

- EMIT «координата объекта» «текст» – выдать сигнал от заданного по координате объекта;
- SET_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» – установка связи;
- DELETE_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» – удаление связи;
- SET_CONDITION «координата объекта» «значение состояния» – установка состояния объекта.
- END – завершить функционирование системы (выполнение программы).

Команда END присутствует обязательно.

Если координата объекта задана некорректно, то соответствующая операция не выполняется и с новой строки выдается сообщение об ошибке.

Если не найден объект по координате:

```
Object «координата объекта» not found
```

Если не найден целевой объект по координате:

```
Handler object «координата целевого объекта» not found
```

Пример ввода:

```
appls_root
/ object_s1 3
/ object_s2 2
/object_s2 object_s4 4
/ object_s13 5
/object_s2 object_s6 6
/object_s1 object_s7 2
endtree
/object_s2/object_s4 /object_s2/object_s6
/object_s2 /object_s1/object_s7
/ /object_s2/object_s4
/object_s2/object_s4 /
end_of_connections
EMIT /object_s2/object_s4 Send message 1
EMIT /object_s2/object_s4 Send message 2
EMIT /object_s2/object_s4 Send message 3
EMIT /object_s1 Send message 4
END
```

1.2 Описание выходных данных

Первая строка:

```
Object tree
```

Со второй строки вывести иерархию построенного дерева.

Далее, построчно, если отработал метод сигнала:

```
Signal from «абсолютная координата объекта»
```

Если отработал метод обработчика:

```
Signal to «абсолютная координата объекта»   Text: «переданная строка»
```

Пример вывода:

```
Object tree
appls_root
    object_s1
        object_s7
    object_s2
        object_s4
        object_s6
    object_s13
Signal from /object_s2/object_s4
Signal to /object_s2/object_s6 Text:  Send message 1 (class: 4)
Signal to / Text:  Send message 1 (class: 4)
Signal from /object_s2/object_s4
```

```
Signal to /object_s2/object_s6 Text:  Send message 2 (class: 4)
Signal to / Text:  Send message 2 (class: 4)
Signal from /object_s2/object_s4
Signal to /object_s2/object_s6 Text:  Send message 3 (class: 4)
Signal to / Text:  Send message 3 (class: 4)
Signal from /object_s1
```

2 МЕТОД РЕШЕНИЯ

Для решения задача было добавлено/изменено:

Класс cl_base:

- Свойства/поля: нет изменений
- Методы:
    - void set_connection(TYPE_SIGNAL p_signal, cl_base* p_object, TYPE_HANDLER p_ob_handler) - создание связи между объектами, модификатор доступа public
    - void delete_connection(Указатель на метод Signal класса Base,Указатель на объект класса Base, Указатель на метод handler класса Base), удаление связи между объектами, модификатор доступа public
    - void emit_signal (TYPE_SIGNAL p_signal, string& s_command) - Выдача сигнала , модификатор доступа public
    - string get_address() полученяи указателя на объект по его абсолютному адресу в дереев иерархии объектов, модификатор доступа public

Класс cl_application:

- Свойства/поля: нет изменений
- Методы:
    - void signal(string s_command) - метод Вывода сигнала, модификатор доступа public
    - void handler(string s_command) - метод обработчика сигнала , модификатор доступа public

3 ОПИСАНИЕ АЛГОРИТМОВ

Согласно этапам разработки, после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

3.1 Алгоритм функции main

Функционал: основная программа.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 1.

Таблица 1 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта ob_cl_application класса cl_application. В качестве параметра конструктору передается значенине nullptr	2
2		Вызов метода build_tree_objects для объекта ob_cl_applicatioin	3
3		Вызов метода exec_app для объекта ob_cl_application	∅

3.2 Алгоритм метода set_connection класса cl_base

Функционал: создание связи между объектами.

Параметры: p_signal - указатель на метод сигнала, p_object - указатель

нацелевой объект, p_ob_handler - указатель на метод обработчика.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода set_connection класса cl_base

№	Предикат	Действия	№ перехода
1		Объявление переменной p_value - указателя на объект типа connection	2
2		Инициализация целочисленной переменной i и присвоение ей значения 0	3
3	Не достигнут конец списка связей		4
			5
4	Полученные параметры совпадают со свойствами текущего объекта объекта из списка связей connections		∅
		Увеличение значения переменной i на 1	3
5		Создание нового объекта типа connection и присвоение его указателю p_value	6
6		Присвоение полю p_signal объекта, на который указывает p_value значения параметра p_signal	7
7		Присвоение полю p_cl_base объекта, на который указывает p_value значения параметра	8

№	Предикат	Действия	№ перехода
		p_object	
8		Присвоение полю p_handler объекта, на который указывает p_value значения параметра p_ob_handler	9
9			∅

3.3 Алгоритм метода delete_connection класса cl_base

Функционал: удаление связи между объектами.

Параметры: p_signal - указатель на метод сигнала, p_object - указатель нацелевой объект, p_ob_handler - указатель на метод обработчика.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода delete_connection класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной i и присвоение ей значения 0	2
2	Не достигнут конец списка связей		3
			∅
3	Полученные параметры совпадают со свойствами текущего объекта объекта из списка связей connections		4

№	Предикат	Действия	№ перехода
		Увеличение значения переменной i на 1	2
4		Удаление связи под номером i из списка связей connections	∅

3.4 Алгоритм метода emit_connection класса cl_base

Функционал: выдача сигнала и отработка соответствующих отработчиков.

Параметры: p_signal - метод сигнала, s_command - строка.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода emit_connection класса cl_base

№	Предикат	Действия	№ перехода
1	Текущий объект готов	Создание переменной p_handler - типа метода обработчика	2
			∅
2		Создание указателя p_object на объект класса cl_base	3
3		Вызов метода, на который указывает p_signal, для объекта, на который указывает p_object. В качестве параметра передается строка s_command	4

№	Предикат	Действия	№ перехода
4		Инициализация целочисленной переменной i и присвоение ей значения 0	5
5	Не достигнут конец списка связей		6
			∅
6	Метод сигнала текущего объекта типа connection из списка связей connections совпадает с p_signal, целевой объект готов	Присвоение переменной p_handler метода обработчика текущего объекта типа connection из списка связей connections	7
			9
7		Присвоение переменной p_object указателя на целевой объект текущего объекта типа connection из списка связей connections	8
8		Вызов метода, на который указывает p_handler, для объекта, на который указывает p_object. В качестве параметра передается строка s_command	9
9		Увеличение значения переменной i на 1	5

3.5 Алгоритм метода handler класса cl_application

Функционал: метод обработчика, методы классы cl_1, cl_2, cl_3, cl_4, cl_5, cl_6 аналогины класса cl_application.

Параметры: s_command.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода handler класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод "\nSignal to ", результата работы метода get_address, вызванногодля текущего объекта, " Text: ", s_command	∅

3.6 Алгоритм метода signal класса cl_application

Функционал: метод сигнала методы классы cl_1, cl_2, cl_3, cl_4, cl_5, cl_6 аналогины класса cl_application.

Параметры: s_command.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода signal класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод "\nSignal from ", результата работы метода get_address,вызванного для текущего объекта	∅

3.7 Алгоритм метода get_address класса cl_base

Функционал: получение адреса объекта в дереве иерархии объектов.

Параметры: нет .

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода get_address класса cl_base

№	Предикат	Действия	№ перехода
1	У текущего объекта нет головного объекта	Возврат "/"	∅
		Инициализация указателя cur на объект класса cl_base и присвоение ему указателя на текущий объект	2
2		Инициализация строковой переменной res и присвоение ей значения ""	3
3	У объекта, на который указывает cur, существует головной объект	Присвоеине переменной res строки "/" + результат работы метода get_name для объекта, на который указывает cur + res	4
			5
4		Присвоеине переменной cur указателя на головной объект объекта, на который указывает cur	3
5		Возврат res	∅

3.8 Алгоритм метода build_tree_objects класса cl_application

Функционал: построение дерева иерархии объектов.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Инициализация строковых переменных name, head_name, sub_name	2
2		Инициализация целочисленной переменной cl	3
3		Инициализация указателя last на объект класса cl_base и присвоеине ему адреса текущего объекта	4
4		Инициализация указателя head на объекта класса cl_base	5
5		Ввод name	6
6		Вызов метода set_name для текущего объекта, в качестве параметра передается name	7
7		Добавление текущего объекта в список all всех объектов дерева иерархии объектов	8
8		Ввод head_name	9
9	headname !+ "endtree"	Ввод значений sub_name, cl	10
			31
10		Присвоение head результата работы метода get_pointer,вызванного для объекта, на который	11

№	Предикат	Действия	№ перехода
		указывает last,в качестве параметра передается head_name	
11	Объект, на который указывает head, не существует	Вывод "Object tree\n"	12
			15
12		Вызов метода get_branch для текущего объекта	13
13		Вывод "\nThe head object ", head_name, " is not found"	14
14		Завершение программы. Код завершеиня 1	∅
15	Подчиненный объект с именем создаваемого объекта уже есть у данного головного объекта	Вывод "Object tree\n"	16
			19
16		Вызов метода get_branch для текущего объекта	17
17		Вывод "\n", head_name, " Dubbing the names of subordinate objects"	18
18		Завершение программы. Код завершеиня 1	∅
19	Номер класса создаваемого		20

№	Предикат	Действия	№ перехода
	объекта дерева иерархии объектов не меньше 2 и не больше 6		
			30
20	Номер класса создаваемого объекта дерева иерархии объектов равен 2	Инициализация объекта класса cl_2 и присвоение его адреса переменной last. В качестве параметов передаются head, sub_name, 2	21
			22
21		Добавление объекта, на который указывает last, в список all всех объектов дерева иерархии объектов	30
22	Номер класса создаваемого объекта дерева иерархии объектов равен 3	Инициализация объекта класса cl_3 и присвоение его адреса переменной last. В качестве параметов передаются head, sub_name, 3	23
			24
23		Добавление объекта, на который указывает last, в список all всех объектов дерева иерархии объектов	30
24	Номер класса создаваемого объекта дерева иерархии объектов равен 4	Инициализация объекта класса cl_4 и присвоение его адреса переменной last. В качестве параметов передаются head, sub_name, 4	25
			26

№	Предикат	Действия	№ перехода
25		Добавление объекта, на который указывает last, в список all всех объектов дерева иерархии объектов	30
26	Номер класса создаваемого объекта дерева иерархии объектов равен 5	Инициализация объекта класса cl_5 и присвоение его адреса переменной last. В качестве параметов передаются head, sub_name, 5	27
			28
27		Добавление объекта, на который указывает last, в список all всех объектов дерева иерархии объектов	30
28	Номер класса создаваемого объекта дерева иерархии объектов равен 6	Инициализация объекта класса cl_6 и присвоение его адреса переменной last. В качестве параметов передаются head, sub_name, 6	29
			30
29		Добавление объекта, на который указывает last, в список all всех объектов дерева иерархии объектов	30
30		Ввод head_name	31
31		Объявление строковых переменных signals_path и handlers_path	32
32		Объявлеине указателя signal на объект класса cl_base и присвоеине ему значения nullptr	33

№	Предикат	Действия	№ перехода
33		Объявлеине указателя handler на объект класса cl_base и присвоеине ему значения nullptr	34
34		Объявление массива signals, содержащего сигналы для классов cl_application, cl_2, cl_3, cl_4, cl_5, cl_6	35
35		Объявление массива handlers, содержащего обработчики для классов cl_application, cl_2, cl_3, cl_4, cl_5, cl_6	36
36		Ввод signals_path	37
37	Значение переменной signals_path не равно "end_of_connections"	Ввод handlers_path	38
			∅
38		Присвоение указателю signal результата работы метода get_address, вызванного для текущего объекта.В качестве параметра передается signals_path	39
39		Присвоение указателю handler результата работы метода get_address, вызванного для текущего объекта. В качестве параметра передается handlers_path	40

№	Предикат	Действия	№ перехода
40		Вызов метода set_connection для объекта, на который указывает signal. В качестве параметров передаются элемент массива signals, номер которого равен классу объекта,на который указывает signal, уменьшенному на 1, handler, элемент массива handlers, номер которого равен классу объекта, на который указывает signal, уменьшенному на 1	41
41		Ввод signals_path	37

3.9 Алгоритм метода exec_app класса cl_application

Функционал: запуск программы.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод "Object tree\n"	2
2		Вызов метода get_branch для текущего объекта	3
3	Не достингут конец списка all всех объеков дерева иерархии объектов	Вызов метода set_status для текущего объекта в списке all всех объеков дерева иерархии объектов	4

№	Предикат	Действия	№ перехода
			5
4		Переход к следующему объекту списка all всех объеков дерева иерархии объектов	3
5		Объявление строковой переменной command	6
6		Ввод command	7
7	Значение переменной command не равно "END"	Объявление массива signals, содержащего сигналы для классов cl_application, cl_2, cl_3, cl_4, cl_5, cl_6	8
			∅
8		Объявление массива handlers, содержащего обработчики для классов cl_application, cl_2, cl_3, cl_4, cl_5, cl_6	9
9		Объявление строковых переменных path1, path2, message	10
10		Объявление целочисленной переменной num	11
11		Объявление указателя signal на объект класса cl_base	12
12		Объявление указателя handler на объект класса cl_base	13
13	Значение переменной	Ввод path1, message	14

№	Предикат	Действия	№ перехода
	command равно "EMIT"		
			16
14		Присвоение указателю signal результата работы метода get_pointer, вызванного для текущего объекта. В качестве параметра передается строка path1	15
15	Объект, который указывает signal, существует	Вызов метода emit_signal для объекта, на который уазывает signal. В качестве параметров передаются элемент массива signals, номер которого равен классу объекта,на который указывает signal, уменьшенному на 1, message	16
		Вывод "\nObject ", path1, "not found"	16
16	command == "set==connect"	Ввод path1, path2	17
			21
17		Присвоение указателю signal результата работы метода get_pointer, вызванного для текущего объекта. В качестве параметра передается строка path1	18
18		Присвоение указателю handler результата работы метода get_pointer, вызванного для текущего	19

№	Предикат	Действия	№ перехода
		объекта. В качестве параметра передается строка path2	
19	Объект, который указывает signal, не существует	Вывод "\nObject ", path1, " not found"	21
			20
20	Объект, который указывает handler, не существует	Вывод "\nHandler object ", path2, " not found"	21
		Вызов метода set_connection для объекта, на который указывает signal. В качестве параметров передаются элемент массива signals,номер которого равен классу объекта, на который указывает signal, уменьшенному на 1,handler, элемент массива handlers, номер которого равен классу объекта, на который указывает signal, уменьшенному на 1	21
21	command == "DELETE_CONNECT"	Ввод path1, path2	22
			27
22		Присвоение указателю signal результата работы метода get_pointer , вызванного для текущего объекта. В качестве параметра передается строка path1	23

№	Предикат	Действия	№ перехода
23		Присвоение указателю handler результата работы метода get_pointer,вызванного для текущего объекта. В качестве параметра передается строка path2	24
24	Объект, который указывает signal, не существует	Вывод "\nObject ", path1, " not found"	27
			25
25	Объект, который указывает handler, не существует	Вывод "\nHandler object ", path2, " not found"	27
			26
26		Вызов метода delete_connection для объекта, на который указывает signal. В качестве параметров передаются элемент массива signals,номер которого равен классу объекта, на который указывает signal, уменьшенному на 1,handler, элемент массива handlers, номер которого равен классу объекта, на который указывает signal, уменьшенному на 1	27
27	command =- "SET_CONNECTION"	Ввод PATH1, NUM	28

№	Предикат	Действия	№ перехода
			30
28		Присвоение указателю signal результата работы метода get_pointer , вызванного для текущего объекта. В качестве параметра передается строка path1	29
29	Объект, который указывает SIGNAL, не существует	Вывод "\nObject ", path1, " not found"	30
		Вызов метода set_reaady для объекта, на который указывает signal. В качестве параметра передается переменная num	30
30		Ввод command	7

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-12.

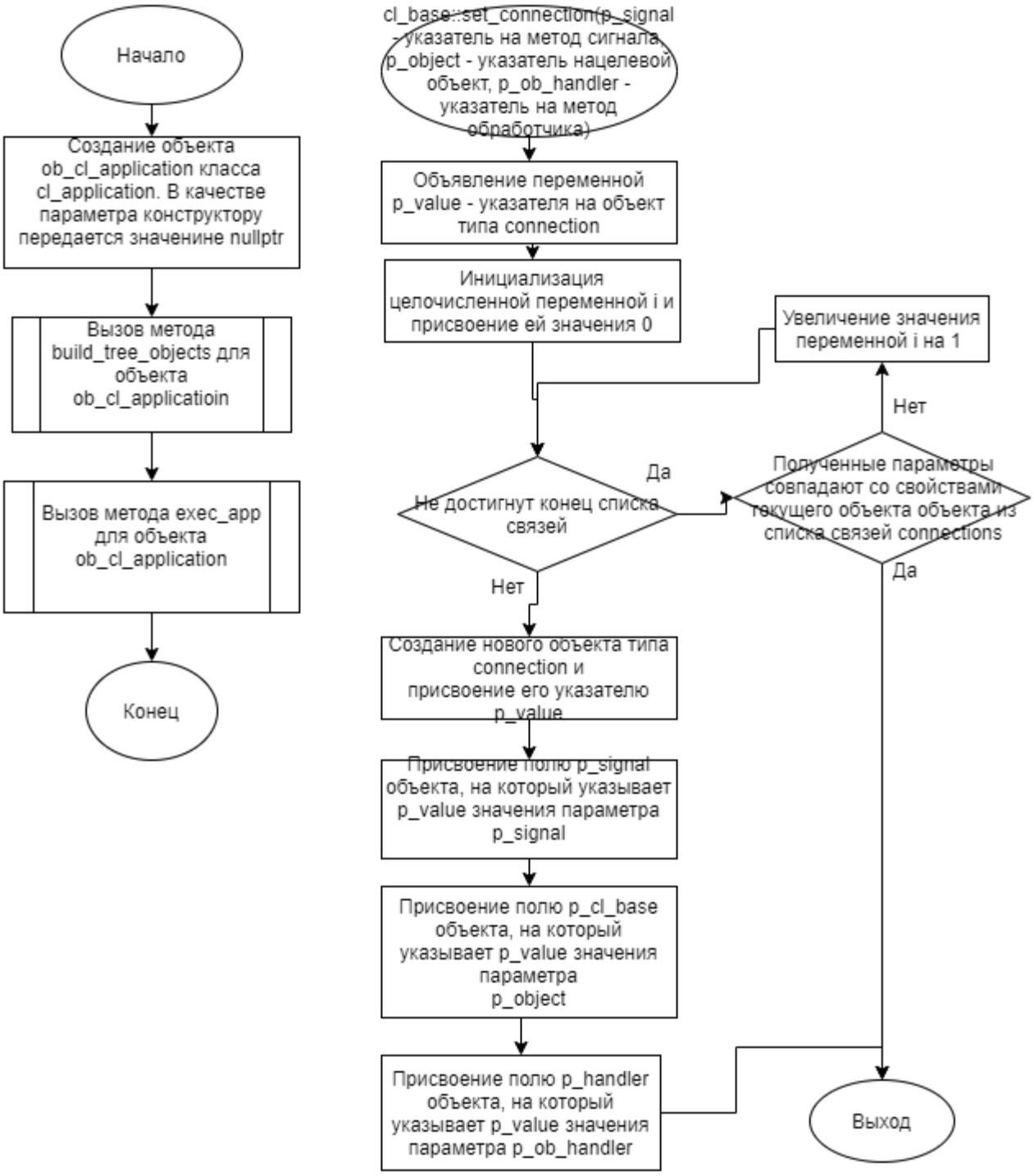

Рисунок 1 – Блок-схема алгоритма

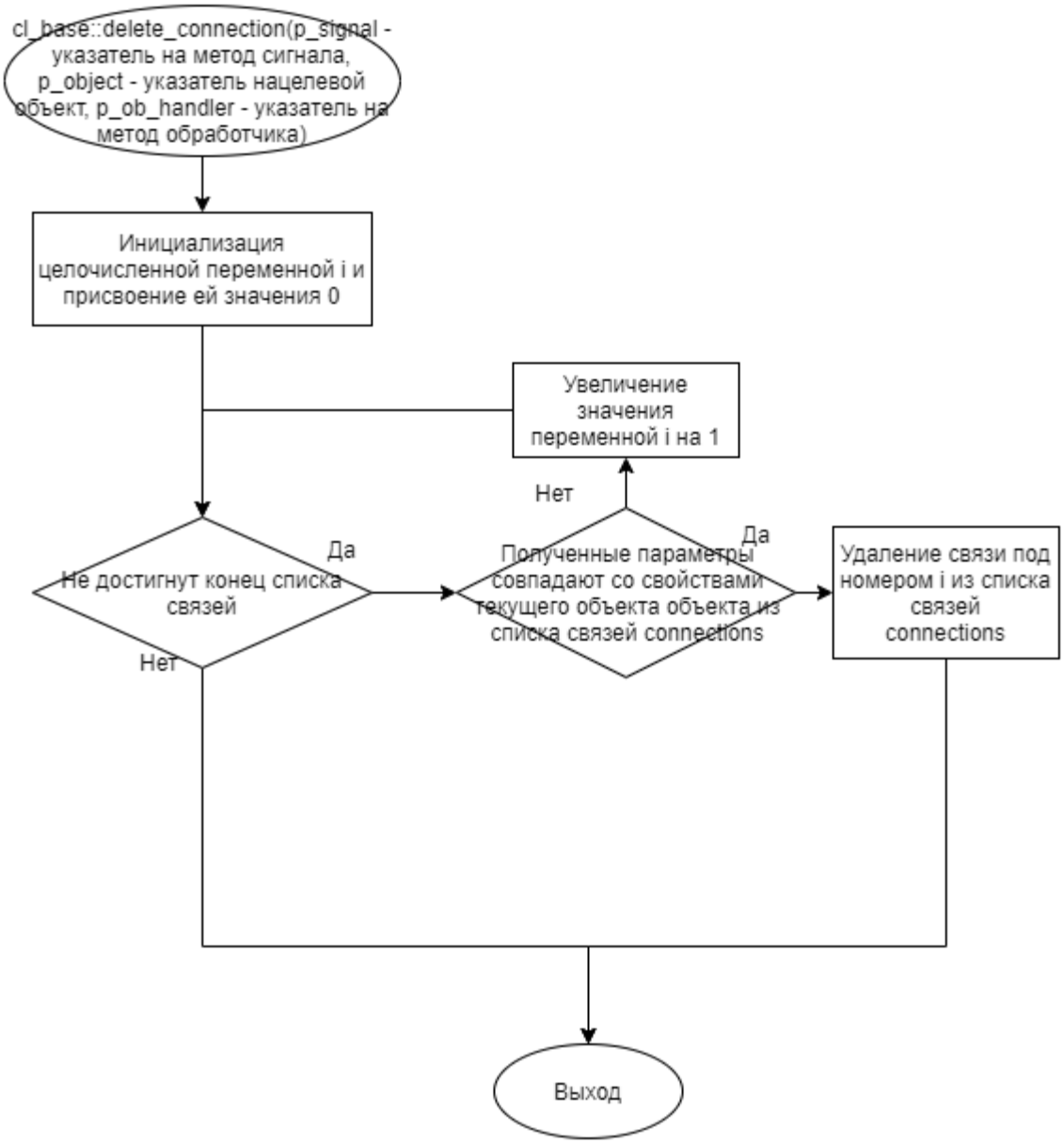

Рисунок 2 – Блок-схема алгоритма

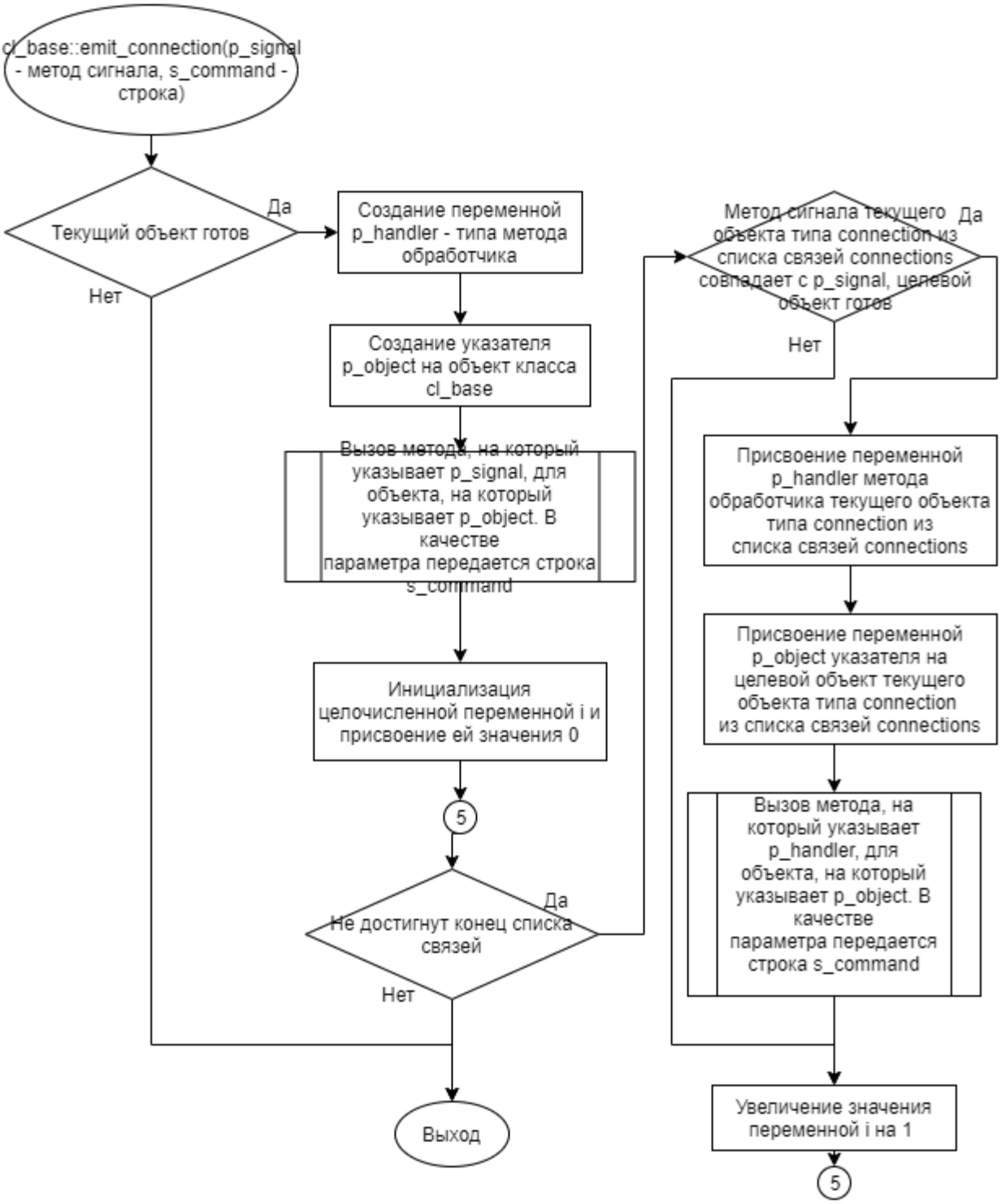

Рисунок 3 – Блок-схема алгоритма

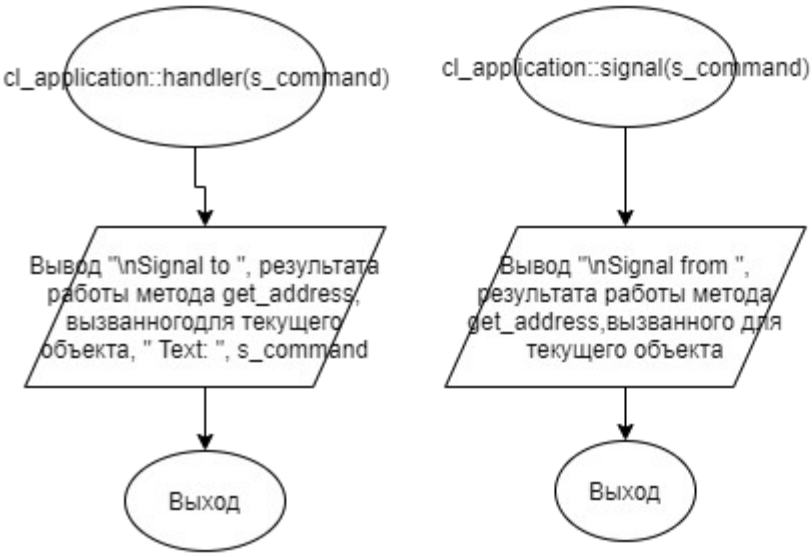

Рисунок 4 – Блок-схема алгоритма

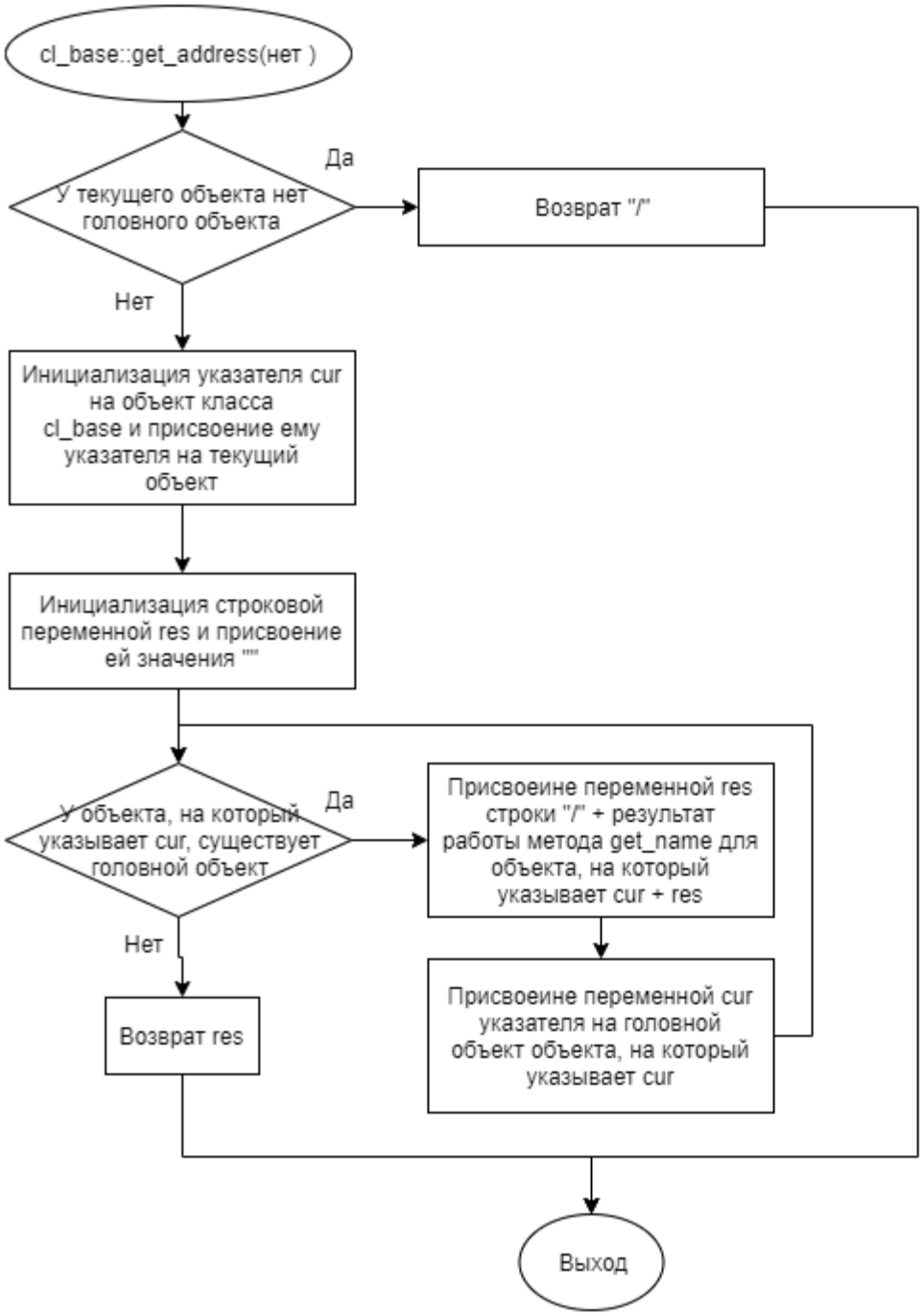

Рисунок 5 – Блок-схема алгоритма

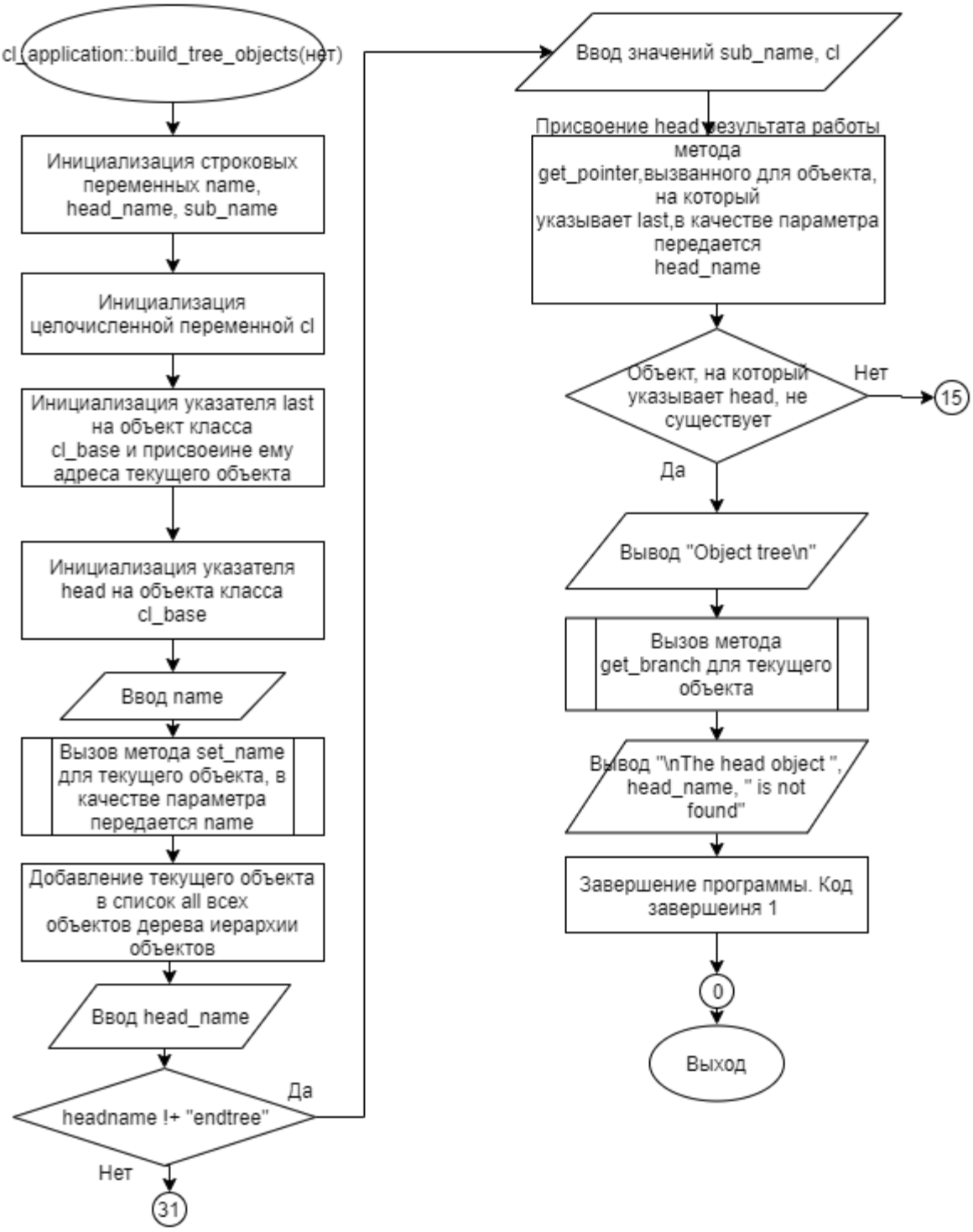

Рисунок 6 – Блок-схема алгоритма

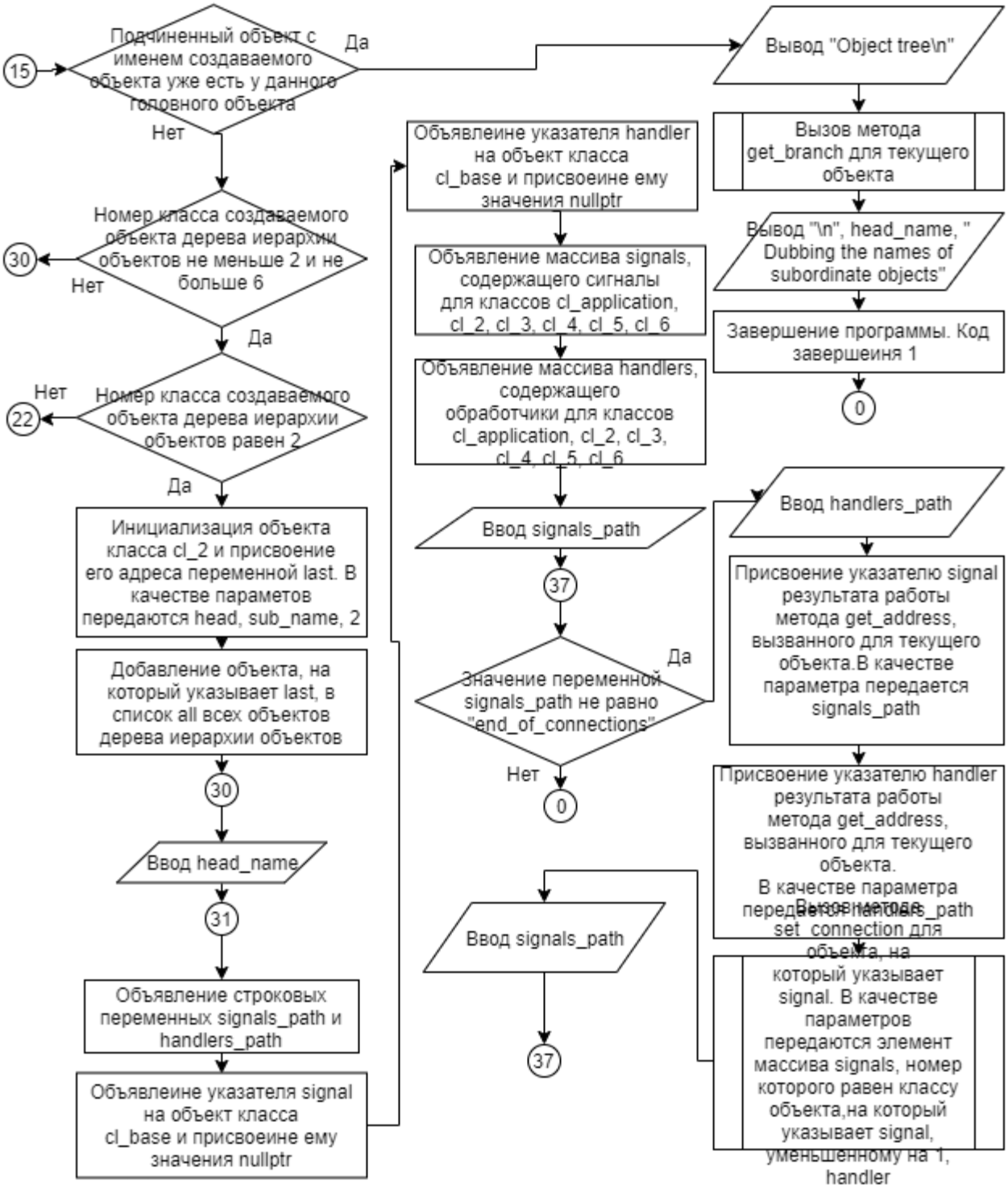

Рисунок 7 – Блок-схема алгоритма

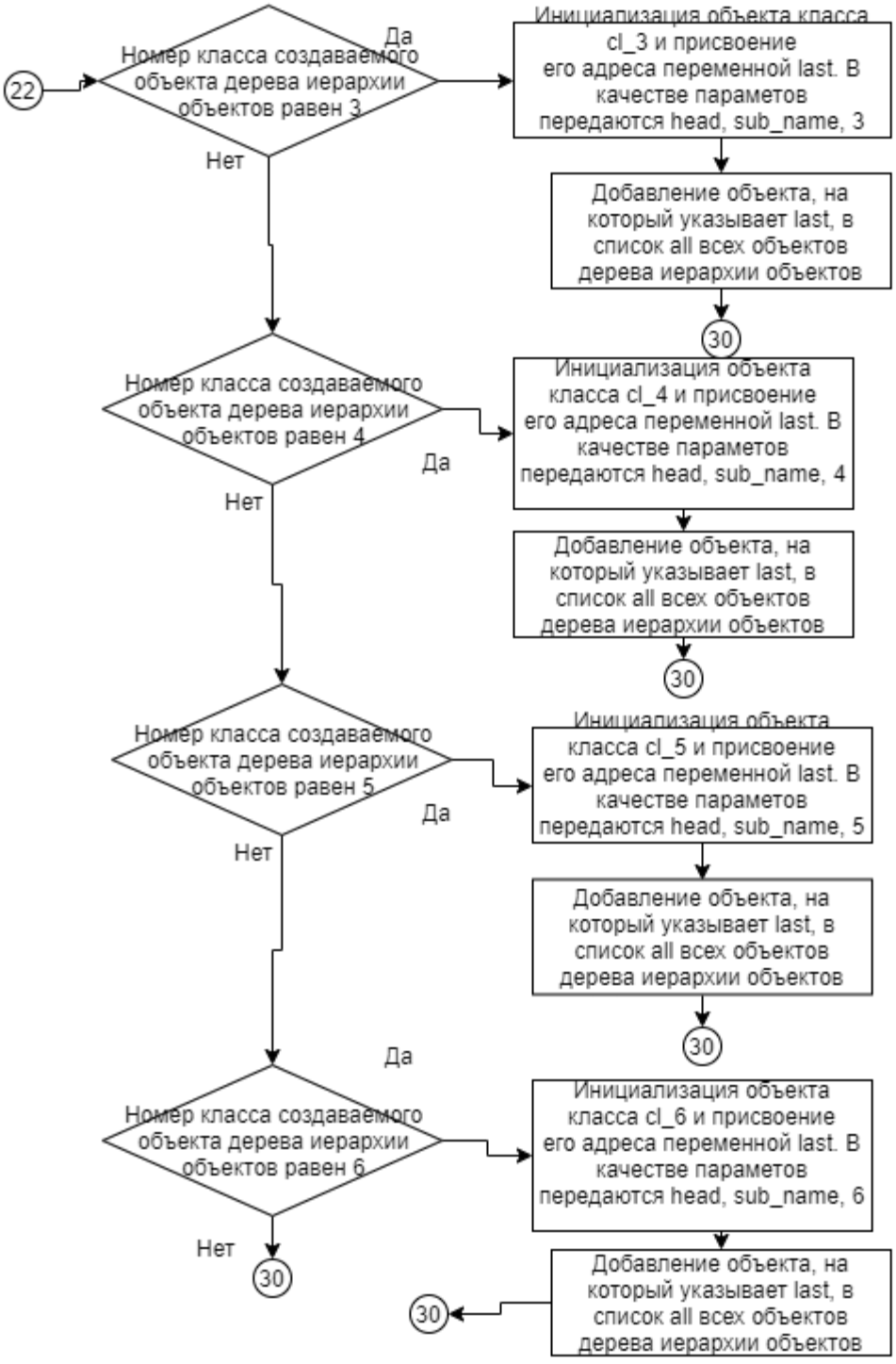

Рисунок 8 – Блок-схема алгоритма

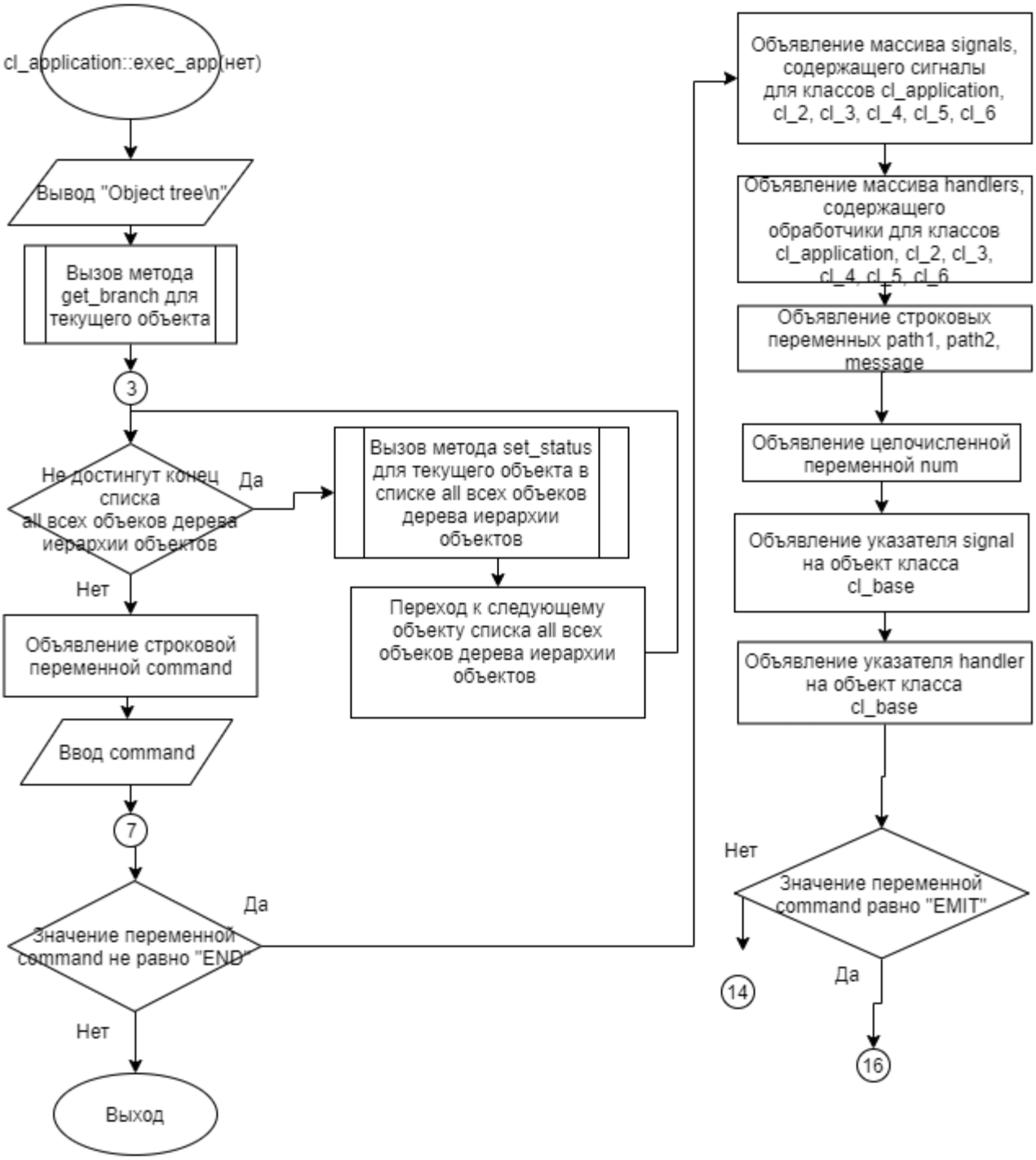

Рисунок 9 – Блок-схема алгоритма

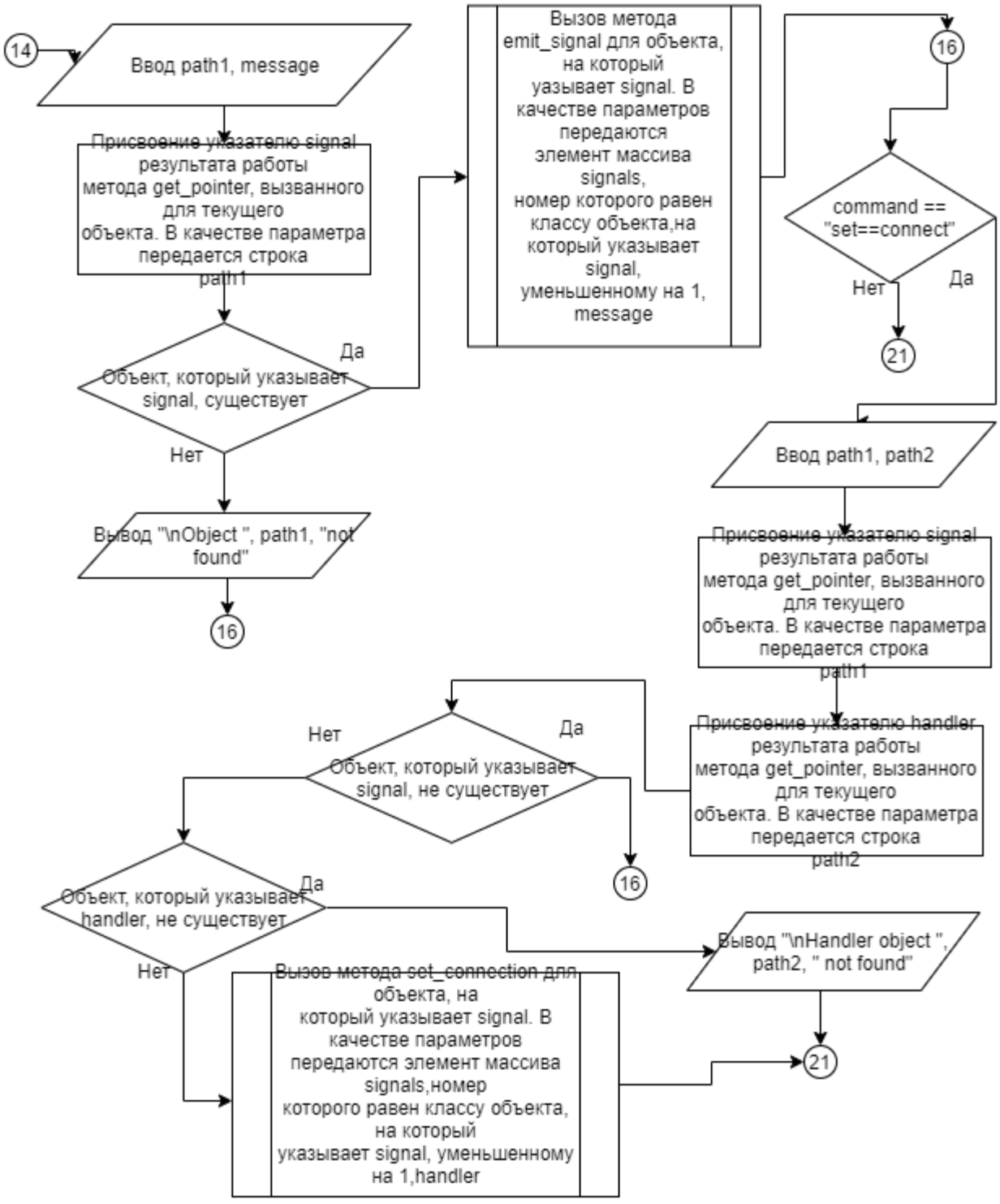

Рисунок 10 – Блок-схема алгоритма

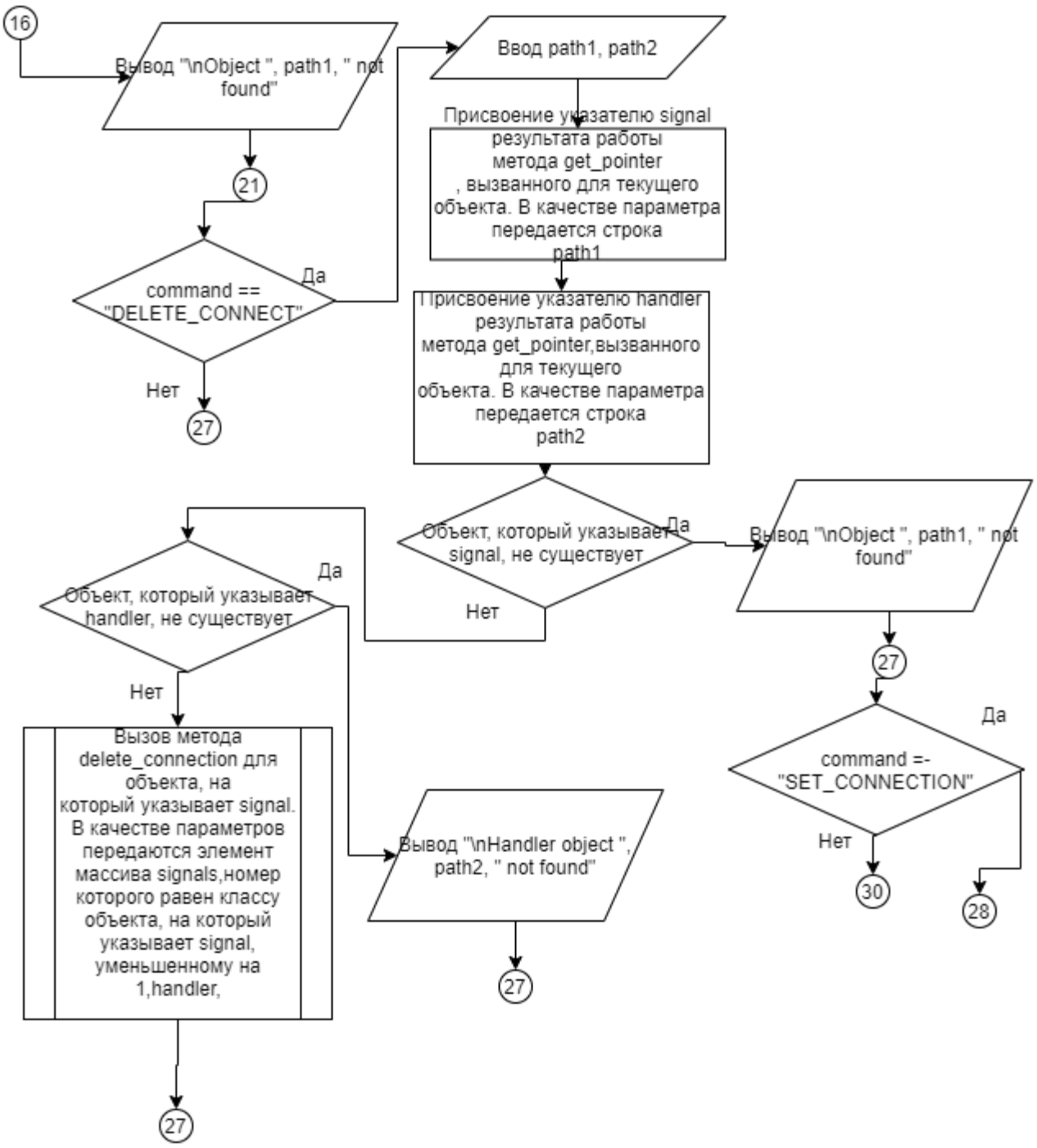

Рисунок 11 – Блок-схема алгоритма

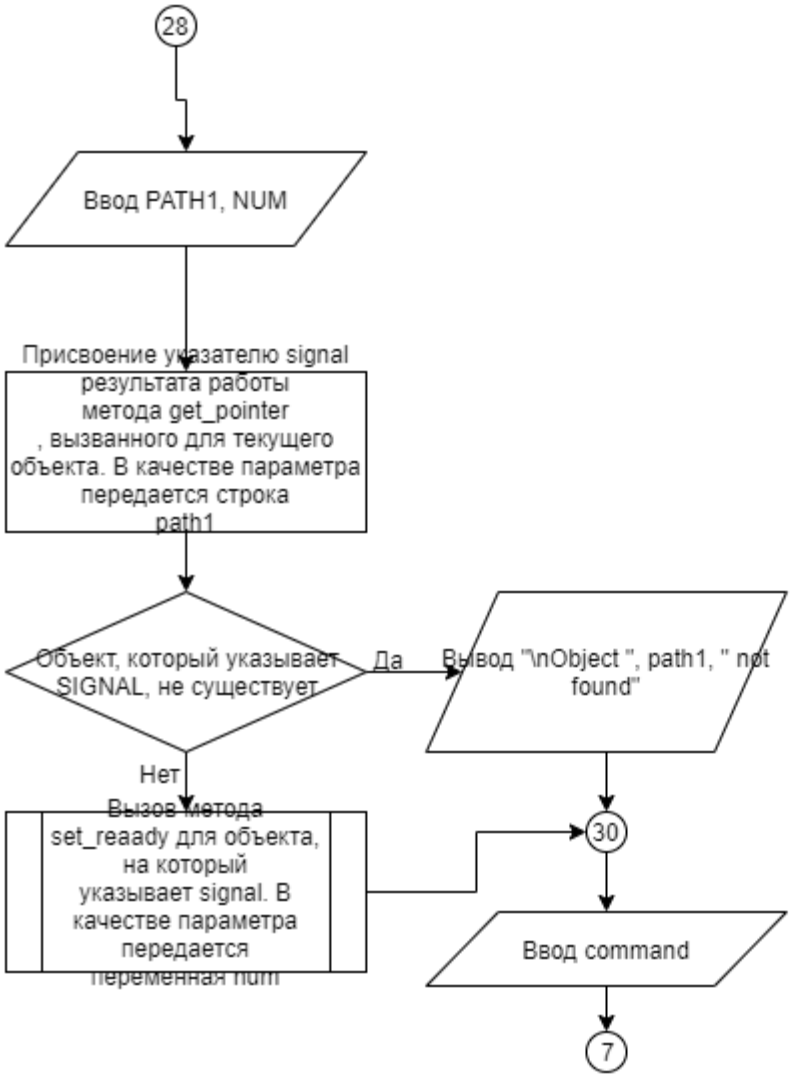

Рисунок 12 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_1.cpp

Листинг 1 – cl_1.cpp

```
#include "cl_1.h"

cl_1::cl_1(cl_base* p_head, string s_name):cl_base(p_head, s_name){}
```

5.2 Файл cl_1.h

Листинг 2 – cl_1.h

```
#ifndef CL_1__H
#define CL_1__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_1: public cl_base{
   public:
      cl_1(cl_base* p_head, string s_name);
};
#endif
```

5.3 Файл cl_2.cpp

Листинг 3 – cl_2.cpp

```
#include "cl_2.h"
#include <iostream>
using namespace std;

cl_2::cl_2(cl_base*  p_head,  string  s_name,  int  cl_num):cl_base(p_head,
```

```
s_name, cl_num){}
void cl_2 :: signal(string &s_command) {
   cout << "\nSignal from " << this -> get_address();
   s_command += " (class: 2)";
}
void cl_2 :: handler(string s_command) {
   cout << "\nSignal to " << this -> get_address() << " Text: " << s_command;
}
```

5.4 Файл cl_2.h

Листинг 4 – cl_2.h

```
#ifndef CL_2__H
#define CL_2__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_2: public cl_base{
   public:
      cl_2(cl_base* p_head, string s_name, int cl_num);
      void signal(string &s_command);
      void handler(string s_command);

};
#endif
```

5.5 Файл cl_3.cpp

Листинг 5 – cl_3.cpp

```
#include "cl_3.h"
#include <iostream>
using namespace std;

cl_3::cl_3(cl_base*  p_head,  string  s_name,  int  cl_num):cl_base(p_head,
s_name, cl_num){}
void cl_3 :: signal(string &s_command) {
   cout << "\nSignal from " << this -> get_address();
   s_command += " (class: 3)";
}
void cl_3 :: handler(string s_command) {
   cout << "\nSignal to " << this -> get_address() << " Text: " << s_command;
}
```

5.6 Файл cl_3.h

Листинг 6 – cl_3.h

```
#ifndef CL_3__H
#define CL_3__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_3: public cl_base{
   public:
      cl_3(cl_base* p_head, string s_name, int cl_num);
      void signal(string &s_command);
      void handler(string s_command);
};
#endif
```

5.7 Файл cl_4.cpp

Листинг 7 – cl_4.cpp

```
#include "cl_4.h"
#include <iostream>
using namespace std;

cl_4::cl_4(cl_base*  p_head,  string  s_name,  int  class_num):cl_base(p_head,
s_name, class_num){}
void cl_4 :: signal(string &s_command) {
   cout << "\nSignal from " << this -> get_address();
   s_command += " (class: 4)";
}
void cl_4 :: handler(string s_command) {
   cout << "\nSignal to " << this -> get_address() << " Text: " << s_command;
}
```

5.8 Файл cl_4.h

Листинг 8 – cl_4.h

```
#ifndef CL_4__H
```

```
#define CL_4__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_4: public cl_base{
   public:
      cl_4(cl_base* p_head, string s_name, int cl_num);
      void signal(string &s_command);
      void handler(string s_command);
};


#endif
```

5.9 Файл cl_5.cpp

Листинг 9 – cl_5.cpp

```
#include "cl_5.h"
#include <iostream>
using namespace std;

cl_5::cl_5(cl_base*  p_head,  string  s_name,  int  cl_num):cl_base(p_head,
s_name, cl_num){}
void cl_5 :: signal(string &s_command) {
   cout << "\nSignal from " << this -> get_address();
   s_command += " (class: 5)";
}
void cl_5 :: handler(string s_command) {
   cout << "\nSignal to " << this -> get_address() << " Text: " << s_command;
}
```

5.10 Файл cl_5.h

Листинг 10 – cl_5.h

```
#ifndef CL_5__H
#define CL_5__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_5: public cl_base{
```

```
   public:
      cl_5(cl_base* p_head, string s_name, int cl_num);
      void signal(string &s_command);
      void handler(string s_command);
};


#endif
```

5.11 Файл cl_6.cpp

Листинг 11 – cl_6.cpp

```
#include "cl_6.h"
#include <iostream>
using namespace std;

cl_6::cl_6(cl_base*  p_head,  string  s_name,  int  class_num):cl_base(p_head,
s_name, class_num){}

void cl_6 :: signal(string &s_command) {
   cout << "\nSignal from " << this -> get_address();
   s_command += " (class: 6)";
}
void cl_6 :: handler(string s_command) {
   cout << "\nSignal to " << this -> get_address() << " Text: " << s_command;
}
```

5.12 Файл cl_6.h

Листинг 12 – cl_6.h

```
#ifndef CL_6__H
#define CL_6__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_6: public cl_base{
   public:
      cl_6(cl_base* p_head, string s_name, int cl_num);
      void signal(string &s_command);
      void handler(string s_command);
};
```

```
#endif
```

5.13 Файл cl_application.cpp

Листинг 13 – cl_application.cpp

```
#include "cl_application.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
#include <cstdlib>
#include <iostream>


using namespace std;

cl_application::cl_application(cl_base*                         p_head_object):
cl_base(p_head_object){}

void cl_application :: signal(string &s_command) {
   cout << "\nSignal from " << this -> get_address();
   s_command += " (class: 1)";
}

void cl_application :: handler(string s_command) {
   cout << "\nSignal to " << this -> get_address() << " Text: " << s_command;
}


void cl_application::build_tree_objects(){
   string name, head_name, sub_name;
   int cl;
   cl_base* last = this;
   cl_base* head;
   cin >> name;
   this -> set_name(name);
   this -> all.push_back(this);
   cin >> head_name;
   while (head_name != "endtree") {
      cin >> sub_name >> cl;
      head = last -> find_object_by_path(head_name);
      if (head == nullptr) {
         cout << "Object tree\n";
         this -> get_branch();
         cout << "\nThe head object "<< head_name << " is not found";
         exit(1);
```

```
      }
      else if (head -> get_sub_object(sub_name) != nullptr) {
         cout << "Object tree\n";
         this -> get_branch();
         cout << "\n" << head_name << "      Dubbing the names ofsubordinate
objects";
         exit(1);
      }
      else if ((cl >= 2) && (cl <= 6)){
         switch (cl) {
            case 2:
               last = new cl_2(head, sub_name, 2);
               this -> all.push_back(last);
               break;
            case 3:
               last = new cl_3(head, sub_name, 3);
               this -> all.push_back(last);
               break;
            case 4:
               last = new cl_4(head, sub_name, 4);
               this -> all.push_back(last);
               break;
            case 5:
               last = new cl_5(head, sub_name, 5);
               this -> all.push_back(last);
               break;
            case 6:
               last = new cl_6(head, sub_name, 6);
               this -> all.push_back(last);
               break;

         }
      }
      cin >> head_name;

   }
   string signals_path, handlers_path;
   cl_base* signal = nullptr;
   cl_base* handler = nullptr;
   TYPE_SIGNAL       signals[]       =       {SIGNAL_D(cl_application       ::
signal),SIGNAL_D(cl_2 :: signal), SIGNAL_D(cl_3 :: signal), SIGNAL_D(cl_4 ::
signal),SIGNAL_D(cl_5 :: signal), SIGNAL_D(cl_6 :: signal)};
   TYPE_HANDLER      handlers[]      =      {HANDLER_D(cl_application      ::
handler),HANDLER_D(cl_2    ::    handler),    HANDLER_D(cl_3    ::    handler),
HANDLER_D(cl_4  ::handler),  HANDLER_D(cl_5  ::  handler),  HANDLER_D(cl_6  ::
handler)};
   cin >> signals_path;
   while (signals_path != "end_of_connections") {
      cin >> handlers_path;
      signal = this -> find_object_by_path(signals_path);
      handler = this -> find_object_by_path(handlers_path);
      signal    ->    set_connection(signals[signal    ->    cl_num    -    1],
handler,handlers[handler -> cl_num - 1]);
      cin >> signals_path;
   }
```

```
}


int cl_application::exec_app(){
   cout << "Object tree\n";
   this -> get_branch();
   for (auto &object: this -> all) {
      object -> set_status(1);
   }
   string command;
   cin >> command;
   while (command != "END") {
      TYPE_SIGNAL      signals[]      =      {SIGNAL_D(cl_application      ::
signal),SIGNAL_D(cl_2 :: signal), SIGNAL_D(cl_3 :: signal), SIGNAL_D(cl_4 ::
signal),SIGNAL_D(cl_5 :: signal), SIGNAL_D(cl_6 :: signal)};
      TYPE_HANDLER     handlers[]     =     {HANDLER_D(cl_application     ::
handler),HANDLER_D(cl_2    ::    handler),    HANDLER_D(cl_3    ::    handler),
HANDLER_D(cl_4  ::handler),  HANDLER_D(cl_5  ::  handler),  HANDLER_D(cl_6  ::
handler)};
      string path1, path2, message;
      int num;
      cl_base* signal = nullptr;
      cl_base* handler = nullptr;
      if (command == "EMIT") {
         cin >> path1;
         getline(cin, message);
         signal = this -> find_object_by_path(path1);
         if (signal != nullptr) {
            signal -> emit_signal(signals[signal -> cl_num - 1],message);
         }
         else {
            cout << "\nObject " << path1 << " not found";
         }
      }
      else if (command == "SET_CONNECT") {
         cin >> path1 >> path2;
         signal = this -> find_object_by_path(path1);
         handler = this -> find_object_by_path(path2);
         if (signal == nullptr) {
            cout << "\nObject " << path1 << " not found";
         }
         else if (handler == nullptr) {
            cout << "\nHandler object " << path2 << " not found";
         }
         else {
            signal -> set_connection(signals[signal -> cl_num - 1], handler,
handlers[handler -> cl_num - 1]);
         }
      }
      else if (command == "DELETE_CONNECT") {
      cin >> path1 >> path2;
         signal = this -> find_object_by_path(path1);
         handler = this -> find_object_by_path(path2);
         if (signal == nullptr) {
            cout << "\nObject " << path1 << " not found";
```

```
            }
            else if (handler == nullptr) {
               cout << "\nHandler object " << path2 << " not found";
            }
            else {
               signal   ->   delete_connection(signals[signal   ->   cl_num-   1],
handler, handlers[handler -> cl_num - 1]);
            }
         }
         else if (command == "SET_CONDITION") {
            cin >> path1 >> num;
            signal = this -> find_object_by_path(path1);
            if (signal == nullptr) {
               cout << "\nObject " << path1 << " not found";
            }
            else {
               signal -> set_status(num);
            }
         }
         cin >> command;
      }
      return 0;
   }
```

5.14 Файл cl_application.h

Листинг 14 – cl_application.h

```
   #ifndef CL_APPLICATION__H
   #define CL_APPLICATOIN__H
   #include <iostream>
   #include "cl_base.h"
   #include "cl_1.h"
   #include "cl_2.h"
   #include "cl_3.h"
   #include "cl_4.h"
   #include "cl_5.h"
   #include "cl_6.h"
   using namespace std;

   class cl_application: public cl_base{
   public:
      vector <cl_base*> all;
      cl_application(cl_base* head= nullptr);
      void signal(string &s_command);
      void handler(string s_command);
      void build_tree_objects();
      int exec_app();

   };
```

```
#endif
```

5.15 Файл cl_base.cpp

Листинг 15 – cl_base.cpp

```
#include "cl_base.h"

cl_base::cl_base(cl_base* p_head_object, string s_name, int cl_num){
   this->p_head_object = p_head_object;
   this->s_name = s_name;
   this->cl_num = cl_num;
   if(p_head_object!= nullptr) {
      p_head_object -> p_sub_objects.push_back(this); // добавление в состав
подчиненных головного объекта
   }
}

cl_base::~cl_base(){
   for (auto p_sub_object: this -> p_sub_objects) {
      delete p_sub_object;
}
}
bool cl_base::set_name(string s_new_name){
   if       (this->p_head_object       &&       this->       p_head_object-
>get_sub_object(s_new_name))
   {
      return false;
   }
   this->s_name = s_new_name;
   return true;
}

string cl_base::get_name(){
   return this->s_name;
}

cl_base* cl_base::get_head_object(){
   return this->p_head_object;
}

void cl_base:: print_tree(){
   cl_base* cur_node = this;
   while (cur_node -> get_head_object() != nullptr) {
      cur_node = cur_node -> get_head_object();
   }
   cout << cur_node -> get_name();
   while (cur_node -> p_sub_objects.size() != 0) {
      int n = cur_node -> p_sub_objects.size();
      cout << "\n" << cur_node -> get_name();
```

```
            for (int i = 0; i < n; i++) {
               cout << "    " << cur_node -> p_sub_objects[i]->get_name();
            }
            cur_node =cur_node -> p_sub_objects[n - 1];
         }
      }

      cl_base* cl_base::get_sub_object(string S_name){
         for (auto sub: p_sub_objects) {
               if (sub->get_name() == S_name) {
                  return sub;
               }
            }
         return nullptr;
      }


      cl_base* cl_base::search_branch(string name){
         if (name == this -> get_name()) {
            return this;
         }
         cl_base* result = nullptr;
         for (auto sub: this -> p_sub_objects) {
            result = sub -> search_branch(name);
            if (result != nullptr) {
               break;
            }
         }
         return result;
      }

      cl_base* cl_base::search_tree(string s_name){
         cl_base* current = this;
         while (current -> get_head_object() != nullptr) {
            current = current -> get_head_object();
         }
         return current -> search_branch(s_name);
      }

      //get_branch
      void cl_base :: get_branch() {
         string s = "";
         int k = 0;
         cl_base* cur = this;
         while (cur -> get_head_object() != nullptr) {
            cur = cur -> get_head_object();
            k++;
         }
         for (int i = 0; i < k; i++) {
            s += "    ";
         }
         if (this == cur) {
            cout << s << this -> get_name();
         }
         else {
```

```
      cout << "\n" << s << this -> get_name();
   }
   for (auto p_sub_object: this -> p_sub_objects)
   {
      p_sub_object -> get_branch();
   }
}

void cl_base::print_status(int spaces){
   cout << endl;
   for (int i =0; i < spaces; i++)cout<< "    ";
   cout << s_name << (state ? " is ready" : " is not ready");
   for(auto sub: p_sub_objects){
         sub->print_status(spaces+1);
      }
}


void cl_base::set_status(int num){
   cl_base* cur = this -> p_head_object;
   bool flg = true;
   if (num == 0) {
      this -> state = num;
      for (auto p_subordinate_object: this -> p_sub_objects) {
         p_subordinate_object -> set_status(num);
      }
   }
   else {
      while (cur != nullptr) {
         if (cur -> state == 0) {
            flg = false;
            break;
         }
         cur = cur -> get_head_object();
      }
      if (flg) {
         this -> state = num;
      }
   }

}


void cl_base::remove_sub_object(string name){
   cl_base* obj_to_remove = get_sub_object(name);
   if (obj_to_remove)
   {
      auto    it    =    std::find(p_sub_objects.begin(),p_sub_objects.end(),
obj_to_remove);
      p_sub_objects.erase(it);
      delete obj_to_remove;
   }
}

bool cl_base::change_head(cl_base* new_head){
```

```
   if (!new_head) return false;
   if (!p_head_object) return false;
   if (new_head->get_sub_object(s_name)) return false;
   cl_base* test_head = new_head;
   while (test_head)
   {
      if (test_head == this) return false;
      test_head = test_head->get_head_object();
   }
   auto iterator = find(p_head_object->p_sub_objects.begin(), p_head_object-
>p_sub_objects.end(), this);
   p_head_object->p_sub_objects.erase(iterator);
   p_head_object = new_head;
   p_head_object->p_sub_objects.push_back(this);
   if (!p_head_object->state)
   {
      set_status(0);
   }
   return true;
}


cl_base* cl_base::find_object_by_path(string path){
   /*cl_base* root = this;
   while (root->get_head_object())
   {
      root = root->get_head_object();
   }
   if (path == "/") return root;
   if(path.length() == 0 || path == ".") return this;
   if (path.find("//") != -1)
   {
      string obj_name = path.substr(2);
      return search_tree(obj_name);
   }
   if (path.find(".") != -1){
      string obj_name = path.substr(1);
      return search_branch(obj_name);
   }
   if (path[0] == '/'){
      string absolute_path = path.substr(1);
      return root-> find_object_by_path(absolute_path);
   }
   int slash_index = path.find('/');
   if (slash_index == -1)
   {
      return get_sub_object(path);
   }
   string subordinate_name = path.substr(0, slash_index);
   cl_base* sub = get_sub_object(subordinate_name);
   if (sub){
      string local_path = path.substr(slash_index + 1);
      return sub->find_object_by_path(local_path);
   }
   return nullptr;
```

```
        */

        cl_base* cur;
        string cur_name;
        if (path == "/") {
        cl_base* root = this;
           while (root -> get_head_object() != nullptr) {
              root = root -> get_head_object();
           }
           return root;
        }
        else if (path.substr(0, 2) == "//") {
        cur = this -> search_tree(path.substr(2));
           if (cur != nullptr) {
              cur_name = cur -> get_name();
              cur -> set_name(".");
           if (this -> search_tree(path.substr(2)) == nullptr) {
              cur -> set_name(cur_name);
              return cur;
           }
           cur -> set_name(cur_name);
              return nullptr;
           }
           return nullptr;
        }
        else if (path == ".") {
           return this;
        }
        else if (path.substr(0, 1) == ".") {
           cur = this -> search_branch(path.substr(1));
           if (cur != nullptr) {
              cur_name = cur -> get_name();
              cur -> set_name(".");
              if (this -> search_branch(path.substr(1)) == nullptr) {
                 cur -> set_name(cur_name);
                 return cur;
              }
              cur -> set_name(cur_name);
              return nullptr;
           }
           return nullptr;
        }
        else if (path.substr(0, 1) != "/") {
           if (path.find_first_of("/") == std::string::npos) {
              return this-> get_sub_object(path);
           }
           else{
           return  this-> get_sub_object(path.substr(0,path.find_first_of("/"))) -
      >find_object_by_path(path.substr(path.find_first_of("/") + 1));
           }
        }
        else if (path.substr(0,1) == "/") {
           return           this->           find_object_by_path("/")           -
      >find_object_by_path(path.substr(1));
        }
```

```
   else {
      return nullptr;
   }
}

//get_address
string cl_base :: get_address() {
   if (this -> get_head_object() == nullptr) {
      return "/";
   }
   cl_base* cur = this;
   string res = "";
   while (cur-> get_head_object()){
      res = "/"+ cur->get_name() +res;
      cur = cur -> get_head_object();
   }
   return res;
}



void    cl_base    ::    set_connection(TYPE_SIGNAL    p_signal,    cl_base*
p_object,TYPE_HANDLER p_ob_handler)
{
   connection* p_value;
   for (int i = 0; i < this -> connections.size(); i++) {
      if ((this -> connections[i] -> p_signal == p_signal) &&
         (this -> connections[i] -> p_cl_base == p_object) &&
         (this -> connections[i] -> p_handler == p_ob_handler)) {
         return;
      }
   }
   p_value = new connection;
   p_value -> p_signal = p_signal;
   p_value -> p_cl_base = p_object;
   p_value -> p_handler = p_ob_handler;
   this -> connections.push_back(p_value);
}


void cl_base :: delete_connection(TYPE_SIGNAL p_signal, cl_base* p_object,
TYPE_HANDLER p_ob_handler) {
   for (int i = 0; i < this -> connections.size(); i++) {
      if ((this -> connections[i] -> p_signal == p_signal) &&
         (this -> connections[i] -> p_cl_base == p_object) &&
         (this -> connections[i] -> p_handler == p_ob_handler)) {
         this -> connections.erase(this -> connections.begin() + i);
         return;
      }
   }
}

void cl_base :: emit_signal(TYPE_SIGNAL p_signal, string &s_command) {
   if (this -> state != 0) {
```

```
            TYPE_HANDLER p_handler;
            cl_base* p_object;
            (this ->* p_signal) (s_command);
            for (int i = 0; i < this -> connections.size(); i++) {
               if ((this -> connections[i] -> p_signal == p_signal) &&
                  (this -> connections[i] -> p_cl_base -> state != 0)) {
                  p_handler = this -> connections[i] -> p_handler;
                  p_object = this -> connections[i] -> p_cl_base;
                  (p_object ->* p_handler) (s_command);
               }
            }
         }
      }
```

5.16 Файл cl_base.h

Листинг 16 – cl_base.h

```
      #ifndef CL_BASE__H
      #define CL_BASE__H
      #include <vector>
      #include <string>
      #include <algorithm>
      #include <iostream>
      #include <queue>
      #include <typeinfo>
      using namespace std;


      #define SIGNAL_D(signal_f) (TYPE_SIGNAL) (&signal_f)
      #define HANDLER_D(handler_f) (TYPE_HANDLER) (&handler_f)

      using namespace std;

      class cl_base{
         private:
         cl_base* p_head_object;
         vector <cl_base*> p_sub_objects;
         string s_name;
         int state = 0;
      public:
         typedef void(cl_base :: *TYPE_SIGNAL) (string&);
         typedef void(cl_base :: *TYPE_HANDLER) (string&);
         struct connection { // Структура задания одной связи
            TYPE_SIGNAL p_signal; // Указатель на метод сигнала
            cl_base* p_cl_base; // Указатель на целевой объект
            TYPE_HANDLER p_handler; // Указатель на метод обработчика
         };
         vector <connection*> connections;
         int cl_num = 1;
```

```
    ~cl_base();
    cl_base(cl_base* p_head, string s_name="Base_object", int cl_nu = 1);
    bool set_name(string new_name);
    string get_name();
    cl_base* get_head_object();
    void print_tree();

    cl_base* search_branch(string name);
    cl_base* search_tree(string name);
    void get_branch();
    void print_status(int spaces = 0);
    void set_status(int num);

    cl_base* get_sub_object(string S_name);
    bool change_head(cl_base* p_head);
    void remove_sub_object(string name);
    cl_base* find_object_by_path(string path);

    string get_address();
    void set_connection(TYPE_SIGNAL p_signal, cl_base* p_object, TYPE_HANDLER
p_ob_hendler);
    void    delete_connection(TYPE_SIGNAL    p_signal,    cl_base*    p_object,
TYPE_HANDLER p_ob_hendler);
    void emit_signal(TYPE_SIGNAL p_signal, string& s_command);


};
#endif
```

5.17 Файл main.cpp

Листинг 17 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include "cl_application.h"
using namespace std;

int main()
{
  cl_application ob_cl_application(nullptr);
  ob_cl_application.build_tree_objects();
  return ob_cl_application.exec_app();
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 10.

Таблица 10 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
appls_root / object_s1 3 / object_s2 2 /object_s2 object_s4 4 / object_s13 5 /object_s2 object_s6 6 /object_s1 object_s7 2 endtree /object_s2/object_s4 /object_s2/object_s6 /object_s2 /object_s1/object_s7 / /object_s2/object_s4 /object_s2/object_s4 / end_of_connections EMIT /object_s2/object_s4 Send message 1 EMIT /object_s2/object_s4 Send message 2 EMIT /object_s2/object_s4 Send message 3 EMIT /object_s1 Send message 4 END	Object tree appls_root object_s1 object_s7 object_s2 object_s4 object_s6 object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1	Object tree appls_root object_s1 object_s7 object_s2 object_s4 object_s6 object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы был разработан и реализован механизм сигналов и обработчиков в базовом классе. Этот механизм предоставляет удобный способ организации взаимодействия объектов, позволяя объектам инициировать сигналы и передавать их нескольким обработчикам. Реализованные методы позволяют установить и удалить связь между сигналом объекта и обработчиком, а также выдать сигнал с передачей данных.

Механизм сигналов и обработчиков имеет широкий спектр применений и может быть использован в различных областях программирования. Он упрощает взаимодействие между объектами, обеспечивая гибкость и модульность. Разработанный механизм предоставляет гибкую архитектуру, полезную при создании сложных систем и приложений.

Применение механизма сигналов и обработчиков позволяет эффективно организовывать взаимодействие объектов, упрощает обмен информацией и повышает переиспользуемость кода. Он предоставляет удобный способ передачи сигналов и данных между объектами, а также гибкое управление взаимодействием в рамках системы.

В итоге, разработанный механизм сигналов и обработчиков является мощным инструментом для организации взаимодействия объектов, способствуя созданию более эффективных, модульных и переиспользуемых систем программного обеспечения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratornyh_rabot_3.pdf (дата обращения 05.05.2021).

3. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).